PhD thesis topic:
Type-based security properties assurance in operating systems

Executive summary: We propose to explore using types in operating system source code as a mean to get assurance on security properties. With the rise of memory-safe languages for system programming like Rust, type-based techniques in operating system source have just recently started being investigated to get assurance on functional correctness. With security properties an additional challenge is the need to consider the whole program at once instead of individual functions or modules. It is thus proposed to address three sub-challenges in the thesis: to identify relevant low-level security properties that support global, high-level properties, to study methods to ensure these low-level properties using the type system of the programming language, and finally to explore how to keep guarantees despite interactions with code in memory-unsafe programming languages like C. Proof-of-concept implementations should be done on Rust-based operating systems as well as on operating systems having added support for Rust code, like Linux.
Keywords: Operating systems; Security; Programming languages; Rust.

Director: Frédéric Tronel (CentraleSupélec, team SUSHI) `frederic.tronel@centralesupelec.fr`
Co-advisors:

- Louis Rilling (DGA, team SUSHI) `louis.rilling@irisa.fr`
- Guillaume Hiet (CentraleSupélec, team SUSHI) `guillaume.hiet@centralesupelec.fr`

Inria team: SUSHI.
Laboratory: IRISA, Rennes (head: Guillaume Gravier – `guig@irisa.fr`).
Location: CentraleSupélec, Rennes campus.
Funding: Fully funded under the Inria/ANSSI framework agreement.

Required skills: System programming.
Appreciated but not mandated skills: Deep understanding of OSes; Programming in Rust.

Context and Description

Operating systems (OS), especially their kernel, are critical software to build applications aiming at providing security properties. A vulnerability in the kernel opens a door for attackers to bypass the application logic, whatever the correctness of the application itself.

Getting assurance on security properties of OS services is known to incur high costs. This traditionally relies on careful design and heavy testing, with costs of up to 400% of a non-secure development effort [1]. While formal verification remains a challenge for this class of software, few projects achieved this goal for an up to 1000% cost [2].

We propose to explore a middle way to get assurance, relying on the compiler to check both of memory safety and higher-level, logical, security properties that should be encoded in the source code using types. It is expected that this reduces the debugging effort and it could make formal verification easier.

Among the many type-based techniques that should be considered for OS development, an example is the *typestate* design pattern [3]. This technique gives assurance on the implementation of state machines, by assigning a dedicated type to each state of the state machine and encoding state machine transitions as functions converting from a source state to a target state. Invalid transitions are thus made impossible because the converting functions just do not exist. *Session types* [4] achieve a similar goal for protocols.

Applying type-based techniques, like typestate analysis, on production source code like the Linux kernel shows that the security of such projects could be improved if types were used to encode logical properties [5]. On the other hand, using the type system of the programming language to ensure functional properties is still the topic of ongoing research [6, 7, 8, 9].

Challenge 1: Identify relevant security properties The security properties studied in OSes are traditionally restricted to integrity, isolation between security contexts (memory), information flow enforcement [10], and security policy enforcement [11]. The latter two are high-level and rely on lower-level properties on the subjects and objects managed by the OS. These lower-level properties are good candidates for a type-based approach and should first be identified.

Challenge 2: Ensure the security properties using the type system of the programming language Although using types to achieve security has been known for decades, this practice using the programming language of an OS is hardly explored. Current approaches rely on a separate language and its compiler to write annotations in the original source code and verify that the code satisfies the specified properties [12]. Moreover programming techniques should be explored to address security properties of increasing complexity, notably going from static, bounded sets of contexts (e.g. the two contexts of kernel and userspace memory) to dynamic, unbounded sets (e.g. all the tasks running on Linux).

Challenge 3: Keep guarantees despite interactions with code in memory-unsafe programming languages like C Since legacy OSes are mostly developed in memory-unsafe programming languages, programming techniques like safe wrappers in Rust should be explored to adapt the code incrementally and keep the guarantees obtained with the type system.

During the thesis a strategy of incremental improvements to existing open-source OS projects will be used to address the previous challenges. While studying code-adaptions and type-based techniques several trade-offs will be considered between development and maintenance cost, gained assurance, and possible performance overhead. Opportunities to help formal verification will be considered for follow-up research.

The Rust programming language is especially a good candidate as it is strongly- and statically-typed, by design the compiler checks the memory safety of programs, and it is the basis of several open-source OS projects, including Redox, which explicitly targets security. Moreover successful Rust-implementations were demonstrated for typestates [13] and session types [14] and the Linux kernel has recently got support for kernel code in Rust.

To gradually study use cases of increasing complexity, the thesis work can thus start on the pure-Rust Redox project before considering complex production-grade projects like Linux.

References

[1] Elaine Venson, Xiaomeng Guo, Zidi Yan, and Barry Boehm. Costing secure software development: A systematic mapping study. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. Comprehensive formal verification of an os microkernel. *ACM Trans. Comput. Syst.*, 32(1), feb 2014.

[3] Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, SE-12(1):157–171, 1986.

[4] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In Costas Halatsis, Dimitrios Maritsas, George Philokyprou, and Sergios

Theodoridis, editors, *PARLE'94 Parallel Architectures and Languages Europe*, pages 398–413, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[5] Yuandao Cai, Peisen Yao, Chengfeng Ye, and Charles Zhang. Place your locks well: Understanding and detecting lock misuse bugs. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3727–3744, Anaheim, CA, August 2023. USENIX Association.

[6] Kevin Boos, Namitha Liyanage, Ramla Ijaz, and Lin Zhong. Theseus: an experiment in operating system structure and state management. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1–19, 2020.

[7] Ramla Ijaz, Kevin Boos, and Lin Zhong. Leveraging rust for lightweight os correctness. In *Proceedings of the 1st Workshop on Kernel Isolation, Safety and Verification*, KISV '23, page 1–8, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Vikram Narayanan, Tianjiao Huang, David Detweiler, Dan Appel, Zhaofeng Li, Gerd Zellweger, and Anton Burtsev. RedLeaf: Isolation and communication in a safe operating system. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 21–39. USENIX Association, November 2020.

[9] Abhiram Balasubramanian, Marek S. Baranowski, Anton Burtsev, Aurojit Panda, Zvonimir Rakamari, and Leonid Ryzhyk. System programming in rust: Beyond safety. *SIGOPS Oper. Syst. Rev.*, 51(1):94–99, sep 2017.

[10] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. sel4: From general purpose to a proof of information flow enforcement. In *2013 IEEE Symposium on Security and Privacy*, pages 415–429, 2013.

[11] Jérémy Briffaut, Jean-François Lalande, and Christian Toinard. Formalization of security properties: enforcement for MAC operating systems and verification of dynamic MAC policies. *International Journal On Advances in Security*, 2(4):325–343, December 2009. ISSN: 1942-2636.

[12] Christopher Brown, Adam D. Barwell, Yoann Marquer, Céline Minh, and Olivier Zendra. Type-driven verification of non-functional properties. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming*, PPDP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[13] José Duarte and António Ravara. Taming stateful computations in rust with typestates. *Journal of Computer Languages*, 72:101154, 2022.

[14] Thomas Bracht Laumann Jespersen, Philip Munksgaard, and Ken Friis Larsen. Session types for rust. In *Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming*, WGP 2015, page 13–22, New York, NY, USA, 2015. Association for Computing Machinery.